# The Comprehensive Command-Line Guide for Windows, macOS, and Linux

## A Guide for Beginners, Intermediates, and Hackers

## Table of Contents


- Introduction
- Part 1: The Beginner's Journey
  - 1.1: What is the Command Line?
  - 1.2: Opening the Command Line
  - 1.3: Your First Commands: Navigating the Filesystem
  - 1.4: Creating and Deleting Directories
  - 1.5: Creating and Deleting Files
  - 1.6: Moving and Renaming Files and Directories
  - 1.7: Viewing File Contents
  - 1.8: Getting Help
- Part 2: The Intermediate Path
  - 2.1: Permissions and Ownership
  - 2.2: Searching for Files and Text
  - 2.3: Piping and Redirection
  - 2.4: Managing Processes
  - 2.5: Environment Variables
  - 2.6: Introduction to Shell Scripting
- Part 3: The Hacker's Handbook
  - 3.1: Networking and Connectivity
  - 3.2: System Information and Monitoring
  - 3.3: Advanced Text Manipulation
  - 3.4: Automation and Scheduling
  - 3.5: Advanced Scripting Techniques
  - 3.6: Package Management
- Glossary
- Index

## Introduction

Welcome to the Comprehensive Command-Line Guide! This document is designed to take you from a complete novice to a command-line "hacker." Whether you're using Windows, macOS, or Ubuntu, the command line is an incredibly powerful tool for developers, system administrators, and anyone who wants to have more control over their computer.

This guide is divided into three parts, each corresponding to a different skill level. We'll cover the fundamental commands, how to combine them into powerful workflows, and how to automate tasks with scripts.

Let's begin!

## Part 1: The Beginner's Journey

This section is for those who are new to the command line. We'll start with the basics and build a solid foundation.

### 1.1: What is the Command Line?

The **command-line interface (CLI)** is a text-based way to interact with your computer. Instead of clicking on icons and menus, you type commands to tell the computer what to do. It might seem intimidating at first, but it's often faster and more powerful than a graphical user interface (GUI).

- **Shell:** The program that interprets your commands. Bash is common on Linux and macOS, while Windows uses PowerShell or the Command Prompt.
- **Terminal/Console:** The window where you type your commands.

### 1.2: Opening the Command Line

- **Windows:**

  - **PowerShell:** Right-click the Start button and select "Windows PowerShell" or "Terminal".
  - **Command Prompt:** Press `Win + R`, type `cmd`, and press Enter.
- **macOS:**
  - Open the "Terminal" app, which is in `Applications/Utilities/`.
- **Ubuntu:**
  - Press `Ctrl + Alt + T` or search for "Terminal" in the applications menu.

### 1.3: Your First Commands: Navigating the Filesystem

Let's start by learning how to move around your computer's file system.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **List files and directories** | `ls` | `dir` | Shows the contents of the current directory. |
| **Change directory** | `cd` | `cd` or `chdir` | Moves you to a different directory. |
| **Show current directory** | `pwd` | `cd` or `echo %cd%` | Prints the "present working directory". |

**Examples:**

- **Linux/macOS:**

```
# List files in the current directory
ls -la
```

```
# Move to the "Documents" directory
cd Documents

# Move up one level
cd ..

# Go to your home directory
cd ~
```

- **Windows:**

```
# List files in the current directory
dir

# Move to the "Documents" directory
cd Documents

# Move up one level
cd ..
```

### 1.4: Creating and Deleting Directories

Organizing your files starts with creating directories (folders).

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Create a directory** | `mkdir` | `mkdir` | Creates a new directory. |
| **Remove a directory** | `rmdir` | `rmdir` | Removes an *empty* directory. |

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Remove a directory and its contents** | `rm -r` | `rmdir /s` or `Remove-Item -Recurse` | Removes a directory and everything in it. **Use with caution!** |

**Examples:**

- **Linux/macOS:**

```
# Create a directory named "projects"
mkdir projects

# Create nested directories
mkdir -p projects/web/css

# Remove the empty "css" directory
rmdir projects/web/css

# Remove the entire "projects" directory
rm -r projects
```

- **Windows:**

```
# Create a directory named "projects"
mkdir projects

# Remove the empty "projects" directory
rmdir projects

# Remove a directory and its contents (will ask for confirma
rmdir /s projects
```

### 1.5: Creating and Deleting Files

Now let's work with files.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Create an empty file** | `touch` | `echo. >` or `New-Item` | Creates a new, empty file. |
| **Create a file with content** | `echo "text" >` | `echo "text" >` | Creates a file and adds text to it. |
| **Delete a file** | `rm` | `del` or `erase` | Removes a file. |

**Examples:**

- **Linux/macOS:**

```
# Create an empty file named "notes.txt"
touch notes.txt

# Create a file with some text
echo "Hello, world!" > greeting.txt

# Delete the "notes.txt" file
rm notes.txt
```

- **Windows:**

```
# Create an empty file named "notes.txt"
echo. > notes.txt

# Create a file with some text
echo "Hello, world!" > greeting.txt
```

```
# Delete the "notes.txt" file
del notes.txt
```

### 1.6: Moving and Renaming Files and Directories

The same command is often used for both moving and renaming.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Move/Rename** | `mv` | `move` or `ren` | Moves a file or directory, or renames it. |

**Examples:**

- **Linux/macOS:**

```
# Rename "greeting.txt" to "hello.txt"
mv greeting.txt hello.txt

# Move "hello.txt" to the "projects" directory
mv hello.txt projects/
```

- **Windows:**

```
# Rename "greeting.txt" to "hello.txt"
ren greeting.txt hello.txt

# Move "hello.txt" to the "projects" directory
move hello.txt projects\
```

### 1.7: Viewing File Contents

There are several ways to see what's inside a file.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **View entire file** | `cat` | `type` | Prints the entire file content to the screen. |
| **View file page by page** | `less` or `more` | `more` | Allows you to scroll through a file. |

**Examples:**

- **Linux/macOS:**

```
# View the entire file
cat hello.txt

# View a long file page by page (press 'q' to quit)
less /var/log/syslog
```

- **Windows:**

```
# View the entire file
type hello.txt

# View a long file page by page
more some_long_file.txt
```

### 1.8: Getting Help

When you don't know how to use a command, you can ask for help.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Get help for a command** | `man` or `command --help` | `help command` or `command /?` | Shows the manual for a command. |

**Examples:**

- **Linux/macOS:**

```
# Get the manual for the "ls" command
man ls
```

- **Windows:**

```
# Get help for the "dir" command
help dir
# Or
dir /?
```

## Part 2: The Intermediate Path

Now that you have the basics down, it's time to explore some more advanced concepts that will make you a more effective command-line user.

### 2.1: Permissions and Ownership

In Linux/macOS, files and directories have permissions for the owner, a group, and everyone else. Windows has a more complex system of Access Control Lists (ACLs).

- **Linux/macOS:** Permissions are represented by a string of 10 characters (e.g., `-rwxr-xr--`). The first character indicates the file type (`-` for a file, `d` for a directory). The next nine characters are three sets of three: read (r), write

(w), and execute (x) permissions for the owner, group, and others.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Change permissions** | `chmod` | `icacls` | Modifies the permissions of a file or directory. |
| **Change ownership** | `chown` | `icacls /setowner` | Changes the owner of a file or directory. |

**Examples:**

- **Linux/macOS:**

```
# Make a script executable for the owner
chmod u+x my_script.sh

# Set permissions to read/write for owner, read-only for gro
chmod 644 my_document.txt

# Change the owner of a file to "jane"
sudo chown jane my_file.txt
```

- **Windows:**

```
# Grant the "Users" group read and write access to a file
icacls my_file.txt /grant Users:(RX,WD)

# Deny write access to a file for the "Guests" group
icacls my_file.txt /deny Guests:(W)
```

### 2.2: Searching for Files and Text

Finding things is a common task. The command line provides powerful tools for this.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Search for files** | `find` | `dir /s` or `Get-ChildItem -Recurse` | Searches for files and directories based on various criteria. |
| **Search for text in files** | `grep` | `findstr` or `Select-String` | Searches for lines containing a matching pattern. |

**Examples:**

- **Linux/macOS:**

```
# Find all files ending with ".log" in the current directory
find . -name "*.log"

# Search for the word "error" in all files in the current di
grep "error" *
```

- **Windows:**

```
# Find all files ending with ".log" in the current directory
dir *.log /s

# Search for the word "error" in all files in the current di
findstr "error" *
```

### 2.3: Piping and Redirection

This is where the command line really starts to shine. You can combine commands to create powerful workflows.

- **Piping (|):** Takes the output of one command and uses it as the input for another.
- **Redirection (>, >>, <):** Redirects the input or output of a command to a file.
  - >: Overwrites a file with the output.
  - >>: Appends the output to a file.
  - <: Takes input from a file.

**Examples:**

- **Linux/macOS:**

```
# List all files, and then search for ".txt" files
ls -l | grep ".txt"

# Save the list of files to a file named "file_list.txt"
ls > file_list.txt

# Append a new line to the file
echo "This is a new line" >> file_list.txt
```

- **Windows:**

```
# List all files, and then search for ".txt" files
dir | findstr ".txt"

# Save the list of files to a file named "file_list.txt"
dir > file_list.txt

# Append a new line to the file
echo "This is a new line" >> file_list.txt
```

### 2.4: Managing Processes

You can view and manage the programs and services running on your computer.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **List running processes** | `ps` or `top` | `tasklist` or `Get-Process` | Shows the currently running processes. |
| **Stop a process** | `kill` | `taskkill` or `Stop-Process` | Terminates a running process. |

**Examples:**

- **Linux/macOS:**

```
# List all running processes
ps aux

# Stop a process with a specific process ID (PID)
kill 12345
```

- **Windows:**

```
# List all running processes
tasklist

# Stop a process with a specific process ID (PID)
taskkill /PID 12345
```

### 2.5: Environment Variables

Environment variables are named values that can affect the way running processes will behave.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **List environment variables** | `env` or `printenv` | `set` or `Get-ChildItem Env:` | Shows all environment variables. |
| **Set an environment variable** | `export` | `set` or `$env:` | Creates or modifies an environment variable. |

**Examples:**

- **Linux/macOS:**

```
# Display the value of the PATH variable
echo $PATH

# Create a new environment variable
export MY_VARIABLE="my_value"
```

- **Windows:**

```
# Display the value of the Path variable
echo %Path%

# Create a new environment variable for the current session
$env:MY_VARIABLE = "my_value"
```

### 2.6: Introduction to Shell Scripting

A shell script is a file containing a series of commands. It's a simple way to automate tasks.

- **Linux/macOS (Bash):**

```
#!/bin/bash
# A simple script to greet the user
```

```
echo "Hello, $USER!"
echo "Today is $(date)."
```

Save this as `my_script.sh`, make it executable with `chmod +x my_script.sh`, and run it with `./my_script.sh`.

- **Windows (Batch):**

```
@echo off
REM A simple script to greet the user
echo Hello, %USERNAME%!
echo Today is %DATE%.
```

Save this as `my_script.bat` and run it by typing `my_script.bat` in the command prompt.

## Part 3: The Hacker's Handbook

Welcome to the advanced section. Here, we'll unlock the full power of the command line, turning you into a true power user.

### 3.1: Networking and Connectivity

These commands are essential for diagnosing network issues and interacting with other systems.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Test connectivity** | `ping` | `ping` | Sends packets to a host to see if it's reachable. |
| **Show network configuration** | `ifconfig` or `ip addr` | `ipconfig` | Displays network interface information. |

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Show network connections** | `netstat` | `netstat` | Shows active network connections and listening ports. |
| **Securely connect to another system** | `ssh` | `ssh` (built-in in modern Windows) | Connects to a remote machine over a secure channel. |
| **Securely copy files** | `scp` | `scp` (built-in in modern Windows) | Copies files between hosts on a network. |
| **Transfer data from or to a server** | `curl` or `wget` | `curl` (Invoke-WebRequest in PS) | Downloads files or data from a URL. |

**Examples:**

- **Linux/macOS:**

```
# Ping Google's public DNS server
ping 8.8.8.8

# SSH into a remote server
ssh user@example.com

# Download a file from the web
wget https://example.com/file.zip
```

- **Windows:**

```
# Ping Google's public DNS server
ping 8.8.8.8

# SSH into a remote server
ssh user@example.com

# Download a file from the web
curl -O https://example.com/file.zip
```

### 3.2: System Information and Monitoring

Get detailed information about your system's hardware and software.

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Show system information** | `uname -a` | `systeminfo` | Displays kernel and operating system information. |
| **Show disk usage** | `df -h` | `fsutil volume diskfree` | Shows free and used space on mounted filesystems. |
| **Show file/directory space usage** | `du -h` | `(Get-ChildItem . -Recurse \| Measure-Object -Property Length -Sum).Sum` | Shows the space used by files and directories. |

| Task | Linux/macOS | Windows (PowerShell/cmd) | Description |
|---|---|---|---|
| **Show memory usage** | `free -h` | `wmic OS get FreePhysicalMemory,TotalVisibleMemory` | Displays the amount of free and used memory. |

### 3.3: Advanced Text Manipulation

`sed` and `awk` are powerful tools for processing and manipulating text.

- **`sed` (Stream Editor):** Used for find-and-replace operations on text.
- **`awk`:** A versatile programming language for working with text files.

**Examples (Linux/macOS):**

```
# Replace all instances of "foo" with "bar" in a file
sed 's/foo/bar/g' my_file.txt

# Print the first column of a space-separated file
awk '{print $1}' my_data.txt

# Print all lines containing "error" from a log file, and show ti
cat /var/log/syslog | grep "error" | awk '{print $5}'
```

### 3.4: Automation and Scheduling

Run commands and scripts automatically at specified times.

- **Linux/macOS (`cron`):** The standard for scheduling tasks. The configuration is stored in a `crontab` file.
- **Windows (Task Scheduler):** A graphical tool, but can be controlled from the command line with `schtasks`.

**Examples:**

- **Linux/macOS (`crontab -e` to edit):**

```
# Run my_script.sh every day at 2:30 AM
30 2 * * * /path/to/my_script.sh
```

- **Windows (`schtasks`):**

```
# Create a task to run my_script.bat every day at 2:30 AM
schtasks /create /tn "My Task" /tr "C:\path\to\my_script.bat
```

### 3.5: Advanced Scripting Techniques

Take your scripts to the next level with loops, conditionals, and functions.

- **Bash (Linux/macOS):**

```
#!/bin/bash

# Function to greet the user
greet() {
  echo "Hello, $1!"
}

# Conditional statement
if [ "$USER" == "root" ]; then
  echo "You are root!"
else
  echo "You are not root."
fi

# Loop
for i in {1..5}; do
  echo "Number: $i"
done

greet "Alice"
```

- **PowerShell (Windows):**

```powershell
# Function to greet the user
function Greet($name) {
  Write-Host "Hello, $name!"
}

# Conditional statement
if ($env:USERNAME -eq "Administrator") {
  Write-Host "You are an administrator!"
} else {
  Write-Host "You are not an administrator."
}

# Loop
for ($i = 1; $i -le 5; $i++) {
  Write-Host "Number: $i"
}

Greet "Alice"
```

### 3.6: Package Management

Easily install, update, and remove software from the command line.

| OS | Package Manager | Example Usage |
|---|---|---|
| **Ubuntu** | `apt` | `sudo apt install nodejs` |
| **macOS** | `brew` (Homebrew) | `brew install git` |
| **Windows** | `winget` or `choco` (Chocolatey) | `winget install -e --id Git.Git` |

## Glossary

- **Argument:** A value passed to a command to modify its behavior.
- **Bash:** A popular shell used on Linux and macOS.
- **CLI (Command-Line Interface):** A text-based interface for interacting with a computer.
- **Command:** An instruction given to the computer.
- **Directory:** A folder that can contain files and other directories.
- **Environment Variable:** A named value that can influence the behavior of running processes.
- **Flag/Option:** A type of argument that modifies a command's behavior, often starting with a `-` or `--`.
- **GUI (Graphical User Interface):** A visual interface with icons and menus.
- **Home Directory:** The default directory for a user.
- **Kernel:** The core of an operating system.
- **Package Manager:** A tool for installing, updating, and removing software.
- **Path:** The location of a file or directory in the filesystem.
- **Permissions:** Rules that determine who can read, write, or execute a file.
- **Piping:** Sending the output of one command as the input to another.
- **PowerShell:** An advanced shell for Windows.
- **Redirection:** Diverting the input or output of a command to a file.
- **Root:** The superuser on Linux/macOS with all permissions.
- **Shell:** A program that interprets commands.
- **Script:** A file containing a sequence of commands.
- **Standard Input/Output (stdin, stdout, stderr):** The default input and output channels for commands.
- **Terminal/Console:** The window where you type commands.

## Index


- `apt`: 7
- `awk`: 7
- `brew`: 7
- `cat`: 4
- `cd`: 4
- `choco`: 7
- `chmod`: 6
- `chown`: 6
- `cp`: 4 (Implied in `mv`)
- `cron`: 7
- `curl`: 7
- `del`: 4
- `df`: 7
- `dir`: 4
- `du`: 7
- `echo`: 4
- `env`: 6
- `erase`: 4
- `export`: 6
- `find`: 6
- `findstr`: 6
- `free`: 7
- `grep`: 6
- `help`: 4
- `icacls`: 6
- `ifconfig`: 7
- `ip`: 7
- `ipconfig`: 7
- `kill`: 6
- `less`: 4
- `ls`: 4
- `man`: 4
- `mkdir`: 4
- `more`: 4

- move: 4
- mv: 4
- netstat: 7
- ping: 7
- ps: 6
- pwd: 4
- ren: 4
- rm: 4
- rmdir: 4
- schtasks: 7
- scp: 7
- sed: 7
- set: 6
- ssh: 7
- systeminfo: 7
- taskkill: 6
- tasklist: 6
- top: 6
- touch: 4
- type: 4
- uname: 7
- wget: 7
- winget: 7